**HTTP 1.1**

HTTP 1.1 is one of the versions of Hypertext transfer protocol, that runs on top of the Internet’s TCP/IP suite of protocols. HTTP 1.1 provides faster delivery of web pages than the original HTTP and reduces web traffic. By introducing persistent connections and pipelining, it resolves the problems in previous versions.

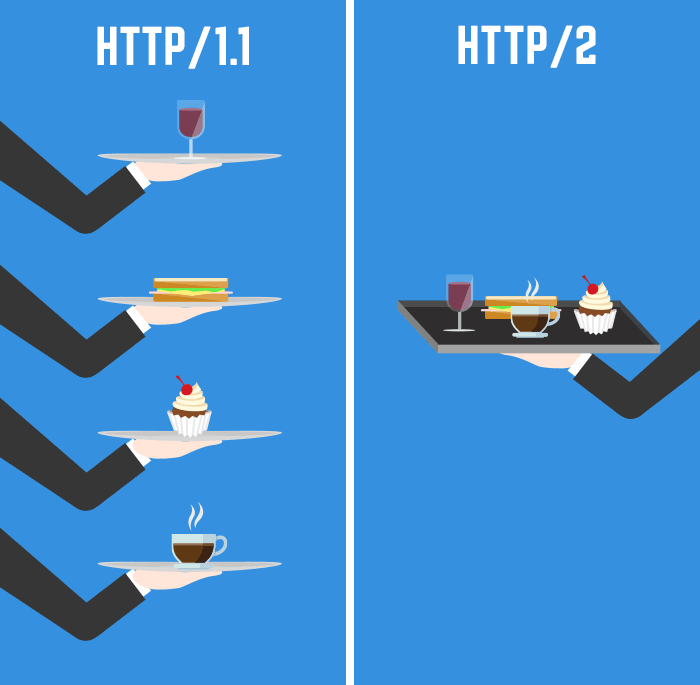
Instead of opening and closing a connection for each application request, HTTP 1.1 provides a persistent connection that allows multiple requests to be batched or pipelined to an output buffer.

**HTTP 2**

HTTP 2 had developed with the intention of reducing web page load latency in HTTP 1.1 by using techniques such as compression, multiplexing, and prioritization. one of the most significant features that distinguishes HTTP/1.1 and HTTP/2 is the binary framing layer, which can be thought of as a part of the application layer in the internet protocol stack.

HTTP 2 still maintaining the same semantics. This ensures that web applications created before HTTP/2 can continue functioning as normal when interacting with the new protocol.

**HTTP 1.1 VS HTTP 2**



* HTTP 1.1 and HTTP 2 ensuring that the requests and responses traveling between the server and client in both protocols reach their destinations as traditionally formatted messages with headers and bodies, using familiar methods.
* HTTP 1.1 transfers these in plain-text messages. So, there is a natural bottleneck to the optimization strategy in it .Multiple data packets cannot pass each other when traveling to the same destination, there are situations in which a request at the head of the queue that cannot retrieve its required resource will block all the requests behind it. This is known as **Head-of-line blocking**. Adding separate, parallel TCP connections could alleviate this issue, but there are limits to the number of concurrent TCP connections possible between a client and server, and each new connection requires significant resources.
* HTTP 2 encodes these into binary, allowing for significantly different delivery model possibilities. In HTTP 2, the binary framing layer encodes requests/responses and cuts them up into smaller packets of information, greatly increasing the flexibility of data transfer.
* HTTP 2 enables a single connection object. The binary framing layer organizes messages into parallel streams of data. Each stream consists of multiple frames. These frame tags are interleaved while transfer and reassembled at end by identifying tags. The interleaved requests and responses can run in parallel without blocking the messages behind them, a process called **multiplexing**. Multiplexing resolves the head-of-line blocking issue in HTTP 1.1. Although the multiplexing inherent in the binary framing layer solves certain issues of HTTP 1.1, multiple streams awaiting the same resource can still cause performance issues.
* To avoid this issue, server can prioritize the responses it is requesting by assigning a weight between 1 and 256 to each stream.it is known as **Stream Prioritization**. The higher number indicates higher priority. In addition to this, the client also states each stream’s dependency on another stream by specifying the ID of the stream on which it depends. If the parent identifier is omitted, the stream is considered to be dependent on the root stream. The server uses this information to create a dependency tree, which allows the server to determine the order in which the requests will retrieve their data. however, that a server may change assigned priorities on its own in certain times stream is blocked from accessing a specific resource.
* In HTTP 1.1, flow control relies on the underlying TCP connection. It follows some **Buffer Overflow** control mechanism based on **ACT Packet**. HTTP 1.1 relies on the transport layer to avoid buffer overflow, each new TCP connection requires a separate flow control mechanism. HTTP 2, however, multiplexes streams within a single TCP connection, and will have to implement flow control in a different manner.
* HTTP 2 multiplexes streams of data within a single TCP connection. As a result, receive windows on the level of the TCP connection are not sufficient to regulate the delivery of individual streams. HTTP 2 solves this problem by allowing the client and server to implement their own flow controls, rather than relying on the transport layer. The application layer communicates the available buffer space, allowing the client and server to set the receive window on the level of the multiplexed streams.
* In HTTP 1.1, if the developer knows in advance which additional resources the client machine will need to render the page, they can use a technique called **Resource Inlining** to include the required resource directly within the HTML document that the server sends in response to the initial get request.
* But there are a few problems with resource inlining. Including the resource in the HTML document is a viable solution for smaller, text-based resources, but for larger files it can ultimately decrease the connection speed and nullify the original advantage gained from using this technique. A major drawback of resource inlining is that the client cannot separate the resource and the document. A finer level of control is needed to optimize the connection, a need that HTTP 2 seeks to meet with server push.
* HTTP 2 enables multiple concurrent responses to a client’s initial get request, a server can send a resource to a client along with the requested HTML page, providing the resource before the client asks for it. This process is called **Server Push**. In this way, an HTTP 2 connection can accomplish the same goal of resource inlining while maintaining the separation between the pushed resource and the document. This means that the client can decide to cache or decline the pushed resource separate from the main HTML document, fixing the major drawback of resource inlining.
* A common method of optimizing web applications is to use **Compression algorithms** to reduce the size of HTTP messages that travel between the client and the server. HTTP 1.1 and HTTP 2 both use this strategy.